The Rockwell 900 series programmable calculators...

...Machines for people.
People for productivity.
Productivity for profit.

Programming Guide

Rockwell

ROCKWELL 900 SERIES
PROGRAMMING GUIDE

INTRODUCTION

This instruction manual has been developed to aid you in learning how to program your Rockwell 900 Series Calculator.

After completion of this manual, you may write some programs which you may want to share with us. If so, we would be happy to hear from you. Please mail your programs to:

Rockwell International
1st DeGrigne Drive
Sunnyvale, California 94086

or

Sumlock Anita/Rockwell International
Anita House, Rockingham Road
Uxbridge, Middlesex, England
UB8 2XL

STANDARD LIBRARY PROGRAMS

Many of the programs that may be required could already be contained in the Rockwell Program Library Listing. Since this listing is frequently updated, an application you have that requires a program could be or will be available in the near future. In either case, contact your Rockwell salesman to see if what you want is available. Your salesman will gladly review a copy with you.

CUSTOM PROGRAMMING

There will be, of course, occasions when you want to have software written specifically for your application. If this is the case, Rockwell has initiated and is continuing to establish, a network of programmers which operate on a local, on-site basis. Our Software Department is maintaining a staff of programmers for this specific purpose. Charges for this service are nominal. Consult your Rockwell salesman for details concerning this service.

TABLE OF CONTENTS

INTRODUCTION TO FLOWCHARTING

A flowchart is a plan, or outline of the way a program should be written. It shows graphically what occurs in a program. By writing a flowchart of a program before you write the actual program steps, you will make your task much simpler. You can organize your thoughts, discover where the difficult parts of the program will be found and decide how to solve them. As a result, when you write the program, you will know exactly how to proceed.

Let's see how the flowcharting technique can be used to outline a situation that occurs every day. Consider the steps involved in selecting a new car. Assume you are going to choose between a Cadillac, a Dodge and a Volkswagen. The following flowchart will outline the steps that you would follow in deciding which car to purchase.

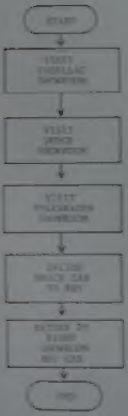


FIGURE A

Notice how this outlines the major tasks and puts them in coherent order. It is possible to re-arrange the tasks somewhat (i.e., by visiting the Dodge showroom before visiting Cadillac), but the basic logic flow cannot be changed. (You couldn't decide which car to buy before visiting at least one showroom).

DEFINITION OF SYMBOLS

The previous flowchart used three symbols that will be applied throughout this book.

They are designed as follows:

1. 

OPERATION SYMBOL
A flowchart breaks down a program into separate operations. A description of each operation is put inside a rectangle called an "Operation Symbol."

2. 

DIRECTION ARROWS
Direction Arrows show the flow directions within a flowchart.

3. 

TERMINAL SYMBOL
Terminal Symbols are used to show where a program begins and ends.

DECISION MAKING

If you study the previous flowchart, you will find that the solution as to which car to buy was not considered. You could make that decision, but how could we break down the problem so that a computer could make the decision?

First, we must establish criteria by which you would decide which car to buy. One logical decision would be based on the price of the car. Our flowchart could be designed so that it compares the price of each car with the amount of money you have to spend, and then selects the best car that you can afford.

The flowchart might look like the flowchart on the next page.

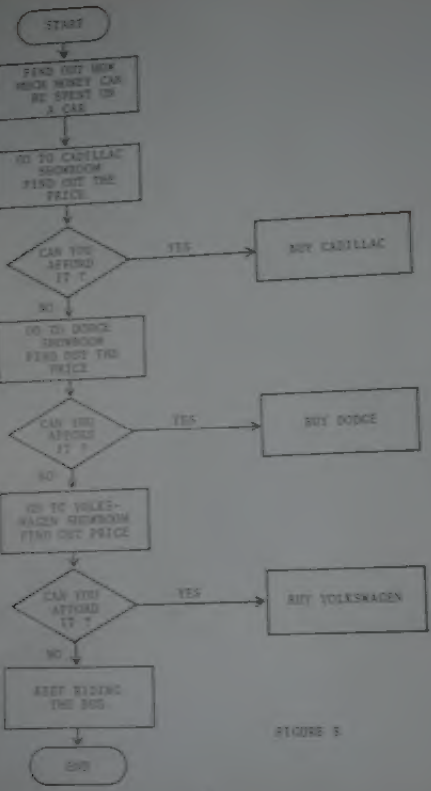

START
FIND OUT HOW MUCH MONEY CAN BE SPENT ON A CAR
GO TO CADILLAC SHOWROOM FIND OUT THE PRICE
CAN YOU AFFORD IT ?
YES
BUY CADILLAC
NO
GO TO DODGE SHOWROOM FIND OUT THE PRICE
CAN YOU AFFORD IT ?
YES
BUY DODGE
NO
GO TO VOLKS-WAGEN SHOWROOM FIND OUT PRICE
CAN YOU AFFORD IT ?
YES
BUY VOLKSWAGEN
NO
KEEP RIDING THE BUS
END

FIGURE 5

The previous flowchart introduced a new symbol called a "Decision Diamond".

The Decision Diamond is used when a decision must be made. For a flowchart (or program) to work properly, each decision must be turned into a question that can be answered with yes or no, or true or false. A Decision Diamond has two direction arrows, one pointing out the path the flowchart will follow if the answer to the question is yes (or true) and the other pointing out the path that flowchart will follow if the answer to the question is no (or false).

LABEL CONNECTORS

CONTINUE TO A

CONTINUE FROM A

Label connectors are used when it is inconvenient, because of the layout or complexity of a flowchart, to show a lengthy direction arrow between two points that are logically connected. An arrow entering a label connector shows where a flowchart is to continue to. An arrow exiting from a connector shows where the program is to continue from. The letters or numbers inside the connector give a name, or label, to the connector.

Sometimes, a flowchart may be written on two or more pages. As a result, a label connector may send you to a place on the page you are reading, or to another page. As a result, two different shapes are used for label connectors.

[illegible] OF EACH LABEL CONNECTOR

Now that label connectors have been explained, we can use them to improve our flowchart. Note that in the flowchart in Figure 1 the three operation symbols contain almost identical operations. One says "Buy Cadillac", another says "Buy Body" and the last says "Buy Volkswagen". These can all be generalized to "Buy Car". This is all the information necessary for the decision making process, since you are already in the proper location when the decision to "Buy Car" is made. We can thus use label connectors to shorten the flowchart. Here is how the new flowchart might look:

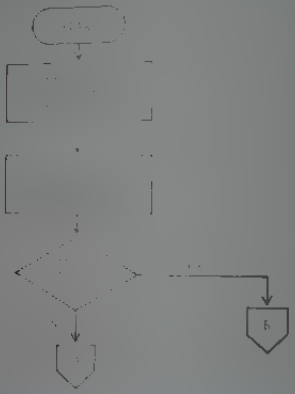


FIGURE 2

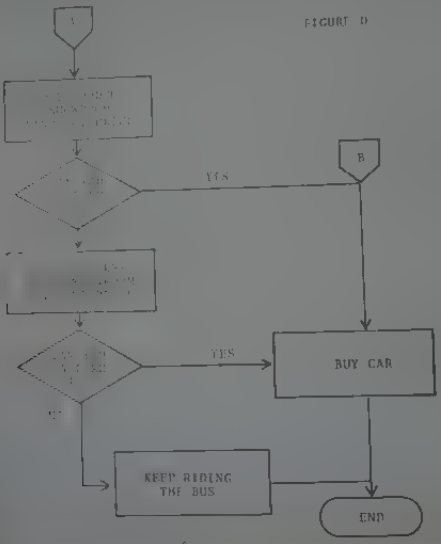
JUMPING

The flowchart in Figure D also illustrates the technique of "jumping". Jumps are used when you are at one point of a flowchart and wish to skip over several operations and go directly to another operation.

When writing programs, you will find that jumps are used in the decision making process.

The continuing flowchart might look as follows.

FIGURE D



Because of the order in which the flowchart takes you to showrooms, it forces you to buy the most expensive car that you can afford. Realistically, buying a car because of the price tag is not a very good idea. The best car is not necessarily the most expensive one. It is the one that best suits your need . . consideration should be given to: styling, prices, safety, performance, seating capacity, trade-in-value, etc. After you've found out this information, you would then buy the most expensive car you could afford that best suits your needs. The flowchart should then be re-written to include these considerations.

First, let's look at [illegible] rule [illegible] flow when writing the flowchart.

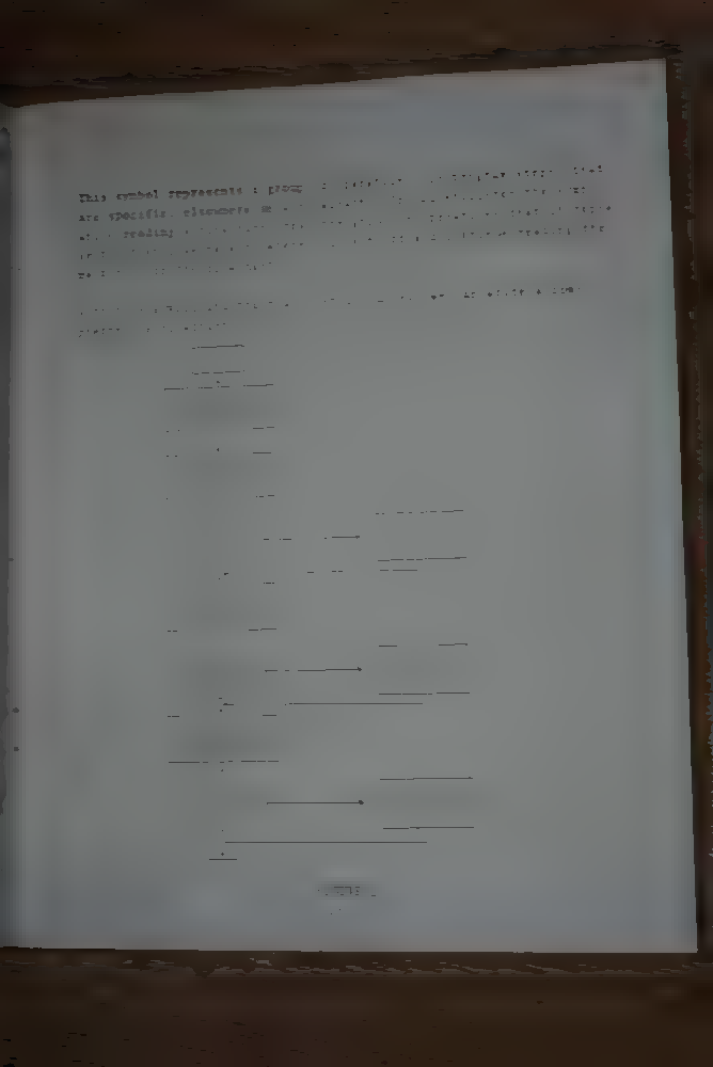
[illegible] the order of expected

1. [illegible]
   [illegible] of the car.
2. [illegible]
   [illegible].
3. [illegible]
   [illegible] other criteria.
4. [illegible]
   [illegible] most showroom.
5. [illegible]
   [illegible].

[illegible] of price, from
[illegible] buy the most
[illegible]

[illegible] introduce a new symbol:

THE DECISION MAKING SYMBOL



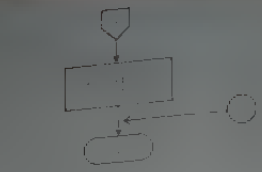


FIGURE 5

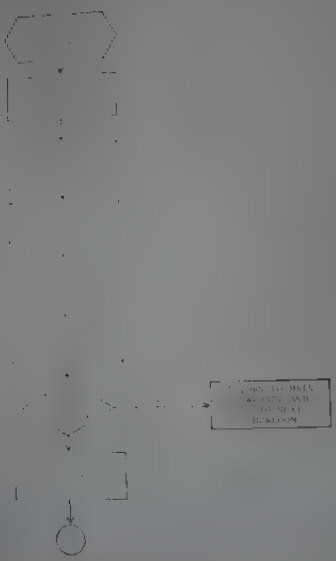


FIGURE 6

SUBROUTINES

This flowchart has introduced an entirely new concept, subroutines. A subroutine is a small flowchart for program that is part of a larger flowchart for program. The subroutine shown here was used to determine if a car met the criteria other than price. If it did, the subroutine directed you to purchase the car. If it didn't, the subroutine directed you to return to the point from which you left the main program. The "return" feature is what distinguishes a subroutine from a loop. A loop takes the program from point A to point B. A subroutine takes the program from point A to point B and then at some predetermined time takes the program back to point A. As illustrated by the flowchart, the program can go to the subroutine from several different parts of a program, but it will always return to the point from which it left. When a computer encounters a branch to a subroutine, it records exactly where it was in the program before it left. As a result, a program can branch to the subroutine at any time and always return to the place it was before it left.

A program is a series of instructions executed automatically by a computing device. The instructions are written by the programmer. To write a program, analyze the best way to solve the problem manually, then execute the problem once, depressing each key in the order in which the problem would be solved. If, to run the problem manually, you had to enter three numbers and execute thirty instructions, a program will reduce your work to entering three numbers on the same key.

[illegible] to write a program to
[illegible] what you want to
[illegible] and have a program
[illegible] the product
[illegible] would like a pro
[illegible]

[illegible] problem manually.

[illegible]

[illegible] variables A,B,C and D

[illegible] solution

A =

B =

C =

D =

To generalize the solution, letters (or variables) were substituted for the numbers 2, 5, 4 and 5.

Now, how would a program be written to compute $\frac{A \times B \times C}{D} = E$? The basic solution was given above, in the solution column.

A x
B x
C
D =

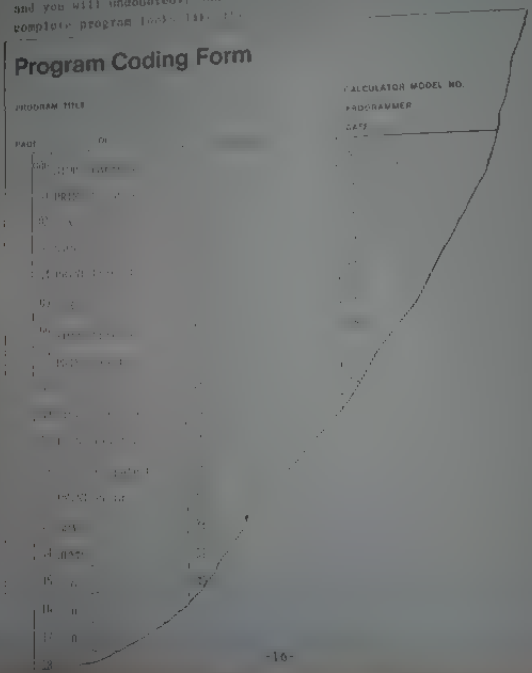
... are additional steps to take in order to enter this program into the calculator. However, the basic idea of how it works is complete.

... these additional steps required for entering a program. ... the program, there are four places where a variable is ... be stored. However, the program cannot be entered touching ... the variables will constantly change. ... to use a certain key that will tell the machine to ... go to the next variable. The instruction that ... is called [STEP] and should be ... the variables in our program.

... the program in fractions are as follows:

[illegible]	EXPLANATION
[illegible]	enter A
[illegible]	enter B
[illegible]	enter C
[illegible]	enter D
[illegible]	compute E

Notice that the last step says "compute E", not "compute and print E". A program will never execute anything unless it is specifically instructed what to do. In this case, it will not print anything unless you instruct it to print. Obviously, you want print the answer and you will undoubtedly want to print the entries as well. The complete program [illegible]



The last line stops
the program at here

Now we're ready to run the program

First, touch the [RUN] key. Then enter the variables in order, touching the [RUN] key after each value. When you're finished, the answer will print automatically.

[illegible]

[illegible] and [illegible]. You [illegible] you should yet [illegible] and see what the [illegible] and run your first [illegible]

[illegible] a program. [illegible] the previous [illegible] to solve it

[illegible] EQUATIONS.

[illegible] you have carefully studied [illegible], choose a method of solving [illegible] for the set of variables. Next, write down the [illegible] that uses a minimum number of key depres-

sions. Finally, print the entries and invert, and insert the instruction [SPACE] as often as desired to set apart entries from answers and to separate different problems.

Let's write a short sample program to illustrate how these steps are followed. A very common business problem is the called for centage increase and decrease. This problem compares the difference and percent of change between periods. For example, if last month's were and this month's are , then there has been an increase of [illegible] over last month.

[illegible]

[illegible]

[illegible] previous period

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible] the amount of change

[illegible] the percentage of change.

[illegible] INSTRUCTIONS.

[illegible]

Program Coding Form

PROGRAM TITLE

CALCULATION MODE NO. PAGE OF

PROGRAMMER

DATE

```
(00) START
     ADD
  02 STOP  Enter current period
     PRINT
  04   +   Add current period
  03 STOP  Enter previous period
  05 PRINT
     JUMP
```

SELECT MEMORY

JUMPS AND DECISIONS

In general, the steps of a program are executed in sequential order. When a [JUMP] instruction is executed, this sequential order changes. The program may go from the 52nd step to the 165th step, or from the 44th step to the 281st step, or from the 392nd step to the 8th step, and so on. The steps in between are ignored. The program simply starts executing from the new address.

Any program step within the memory size of the calculator can be addressed by a Jump instruction.

Jumps fall into two categories, conditional and unconditional. Unconditional jumps simply take a program from one address to an [illegible] For example, if the [illegible] "Jump to Step [illegible] unconditionally. Conditional [illegible] jump from step 28 to 18 only if some [illegible] Therefore, conditional jumps are used when [illegible]

[illegible] different types of decisions. All con- [illegible] register. If we call this number X, [illegible]

CONDITION	INSTRUCTION
[illegible]	Jump if Positive [JUMP] [+]
[illegible]	Jump if Negative [JUMP] [-]
[illegible]	Jump if Zero [JUMP] [=]
[illegible]	Jump if No Entry [JUMP] [AX]

In each case, if the statement is true, the program will jump. If the statement is false, the program will pass through and go to the next step.

To see how jumps are used, write a program to test if X is greater than zero, less than zero, or equal to zero.

1. If X is greater than zero, Print 0 and then print 12345.
2. If X is less than zero, print X and then print a negative 12345.
3. If X equals zero, [illegible] then [illegible].

First analyze the problem [illegible]

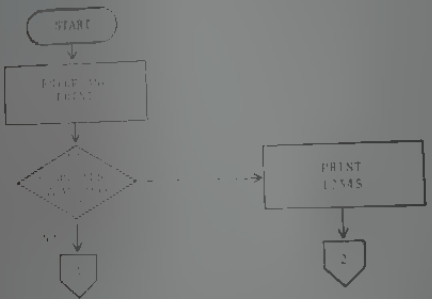


FIGURE 11

Now that we have written the flowchart, we can write the program and insert [STOP], [PRINT] and [GOSUB] instructions. Conditional jumps will be inserted to replace the decision diamonds.

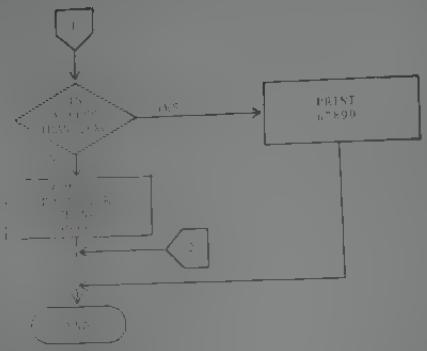


FIGURE 1

Program Coding Form

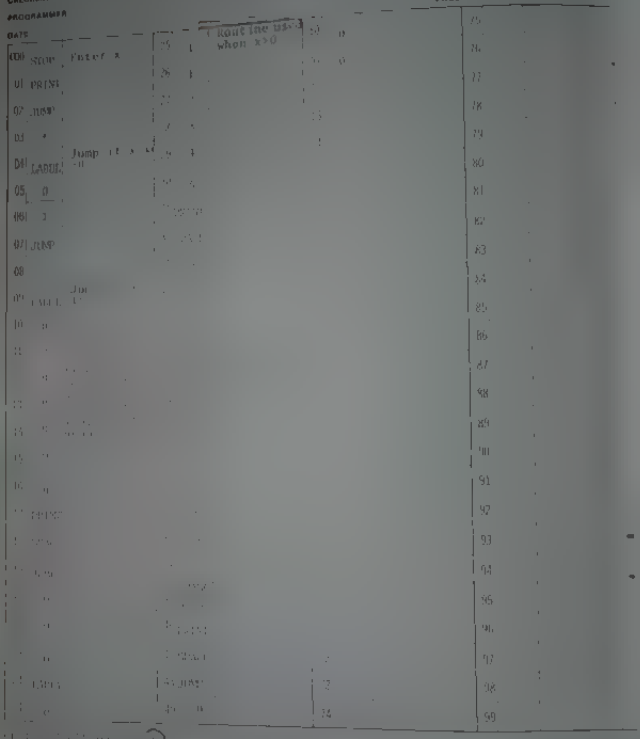
PROGRAM TITLE

CALCULATOR MODEL NO

PROGRAMMER

DATE

PAGE OF



this program used two labels, label 01 and label 02. A label is simply a symbolic address. For more information, please refer to the explanation of [LABEL] in the Advanced Operation Instruction Manual.

This problem involved tests and ranges. However, most programs require that data be processed before it can be tested. For example, write a program in which a number is entered. If X is less than or equal to 100, the program will multiply X by 2 and print the result. If X is greater than 100, the program will divide X by [illegible] and print the result. In either event, after printing the [illegible], the program will [illegible] for a step and prepare to [illegible] another X value.

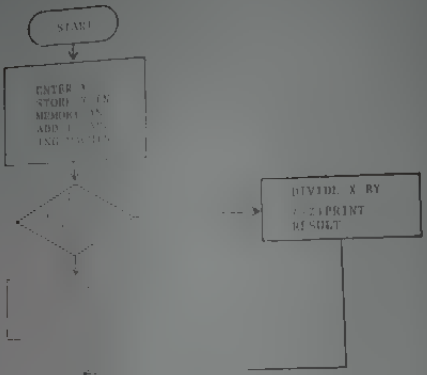
[illegible] the three basic steps to [illegible]

[illegible] the best way to solve it [illegible]

[illegible] the problem, subtract X from 500 to determine [illegible] you either multiply X by (2) or divide X [illegible] depending on the result of the operation. [illegible] from 500 [illegible] the entry we will destroy [illegible] temporarily store X in a memory while performing the test. After the test has been made, we can recall X from the memory and properly process it.

2. Write down the solution. This program involves both decision and calculations, so it is best to construct a flowchart.

[illegible] is to write the solutions to the two routines [illegible] use either 2 times X or X divided by -2.


START
DIVIDE X BY

Program Coding Form

PROGRAM TITLE

CALCULATOR MODEL NO.

PROGRAMMER

DATE

PAGE OF

This program [illegible]

[PROG]. [illegible]

the [illegible]

[illegible]

[illegible] problem

[illegible] are used. Let's put [illegible]

As of this writing, social security [illegible] at the rate of 5.85% of annual earnings, up to [illegible] $[illegible]. After a person's [illegible] social security is no longer deducted.

[illegible]

[illegible] to take [illegible] Consider [illegible]

	Weekly Earnings
	$ [illegible]
	[illegible]
	[illegible]
	[illegible]
	565

[illegible] the social [illegible] doesn't, as [illegible] "C" has earned [illegible] $[illegible] in earnings. [illegible] a portion [illegible] social security and a por- [illegible] program will have to [illegible] are exactly

the opposite of each other. 'P's' weekly earnings will bring the year-to-date earnings up to exactly $13,200. 'P's' year-to-date was exactly $13,200 before the weekly earnings were added.

When writing a program, carefully consider every conceivable combination of circumstances and prepare the program to properly process each potential problem.

Manually solving these problems is easy. For 'A' and 'D', we multiply the weekly earnings by 5.85%. For 'B' and 'C', we do nothing (first case). For 'E', we first find the amount of which social security must be deducted [illegible] year-to-date earnings from $13,200), [illegible] the remainder by 5.85%.

[illegible] problems can be solved

			Explanation
[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	No Deduction
[illegible]	[illegible]	[illegible]	$13,200-YTD=amount on which to pay social security.
[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	No Deduction

As you can see, getting the program to choose the proper solution is necessary. Drawing a flowchart will give us direction for placing decisions. Remember that we are calling the [illegible] 'Y-T-D' and the amount earned in the [illegible]

FIGURE 4

Take a second to look at the question inside the first decision [illegible] another way of asking, "Is YTD greater [illegible] to ask, "Is [illegible] greater than [illegible] the way [illegible] are [illegible] [illegible] must be written with this in mind.

You should also [illegible] the question in the second [illegible] "Is [illegible] greater [illegible] specifically phrased to [illegible] weekly earnings [illegible] exactly [illegible]. If the [illegible] greater than or equal [illegible] would not operate correctly. [illegible] social security [illegible] the need for drawing [illegible] values.

[illegible] JUMP INSTRUCTIONS.

Program Coding Form

PROGRAM TITLE

CALCULATOR MODEL NO.

PROGRAMMER

DATE

PAGE OF

00	CLEAR ALL						
01	STOP	Enter amount earned during pay period (PP)					
02	PRINT						
03		Add PP to memory					
04	STOP	Enter YTD earnings					
05	PRINT						
06		Accumulate YTD					
07							
08							
09							
10							
11							
12							
13							
14							
15							
16	LABEL						
17							
18							
19							
20							
21							
22							
23							
24							
25			0				
26			2				
27			M				
28							
29							
30			5				
31							
32							
33							
34							
35							
36							
37							
38							
39							
40							
41							
42							
43							
44							
45							
46							
47							
48							
49							
50							
51							
52							
53							
54							
55							
56							
57							
58							
59							
60							
61							
62							
63							
64							
65							
66							
67							
68							
69							
70							
71							
72							
73							
74							
75							
76							
77							
78							
79							
80							
81							
82							
83							
84							
85							
86							
87							
88							
89							
90							
91							
92							
93							
94							
95							
96							
97							
98							
99							

[illegible]

There are many programs that can be easily written within the range of [illegible] available in your calculator. [illegible] is rarely [illegible] programs. However, at [illegible] so that you will be [illegible] extensive programs [illegible] are available [illegible]

[illegible] of the keys [illegible] For example, [illegible] INSERT MA [illegible] and [illegible]. [illegible] By contrast, [illegible] all data and [illegible] of keys are [illegible] You [illegible] available. [illegible] that you [illegible] These [illegible] functions [illegible] memories.

[illegible] registers are [illegible] program to see [illegible] takes advantage [illegible] the present key. [illegible] re-writing.

[illegible] to consider:
[illegible] week's pay.
[illegible] portion of the week's pay.
[illegible] at all.

They have been handled by writing three separate routines that handle each situation and then [illegible] back to the beginning of program.

We can save a number of steps by [illegible] parts of the routines, [illegible] than one routine, and letting some instructions [illegible]

Here is a simple example [illegible] but explains the concept [illegible] label [illegible] that prints zero when [illegible] steps are:

[illegible]

Step Number	Key Depression
045	0
046	JUMP
047	0
048	3
049	5

This will have no effect on the answers that print. The program had been printing zero at step 045. Now it will print zero at step 035. In either case, after printing the answer it will jump to step 000.

Using this philosophy, we can shorten the rest of the program,

Program Coding Form

CALCULATOR MODEL NO.

PROGRAM TITLE

PROGRAMMER

DATE

PAGE OF

Step	Key	Comment	Step	Key	Comment	Step	Key	Step	Key
000	CLEAR ALL		25	0	of P	50		75	
01	STOP	Enter amount earned during last pay	26	?		51	0	76	
02	PRINT	period (P)	27			52		77	
03	M+		28	LABEL		53		78	
04	STOP	Enter YTD	29			54		79	
						55		80	
					[illegible]	56		81	
						57		82	
						58		83	
						59		84	
						60		85	
						61		86	
						62		87	
					[illegible] only	63		88	
						64		89	
						65		90	
						66		91	
						67		92	
						68		93	
				LABEL	No soc. sec. to be paid	69		94	
				0		70		95	
		YTD+P-13200	46	1		71		96	
	JUMP	Jump if soc. sec. is to [redacted] paid on P.	47	0	Zero to print	72		97	
23		Continue if soc.	48	JUMP		73		[redacted]	
24	LABEL	sec. is to be paid on portion	[redacted]	0		[redacted]		99	

SELECT MEMORY 1 2 358- 4 5 6 7 8 9

This program uses the same basic logic as the previous one. If you have any questions, please refer to the explanation of the earlier program.

Our 67 step program has been cut down to 62 steps by making some simple changes. It's obviously valuable to cut 15 steps, but the percentage saving, 23%, is even more significant. Theoretically, the same techniques could help you cut a program that first took 67 steps down to 44% steps.

During the rest of this manual we will discuss many other techniques that will help you save steps. However, it is literally impossible to use all techniques on the best thing to do is to constantly look for ways to shorten programs while writing them.

As you will build a mental library of step-saving methods, and while you will find yourself easily solving problems that you have once thought to be far beyond your calculator's capabilities.

A common problem in business statistics is finding the standard deviation of ungrouped data. The formula for the standard deviation (SD) is:

$$SD = \sqrt{\frac{\Sigma X^2}{n} - \left(\frac{\Sigma X}{n}\right)^2}$$

where ΣX is the sum of all variables, ΣX^2 is the sum of the squares of the variables, and n is the number of variables.

[illegible] how easy it is to write [illegible]

[illegible] A good program for standard deviation [illegible] to enter each X-value just [illegible] is made, the program should [illegible] X, then accumulate ΣX^2 and n. [illegible] ready to accept the next [illegible]

[illegible] entered, there should be an easy [illegible] program to jump to a routine that [illegible] data and print the answer.

[illegible] will have different numbers of variables. [illegible] would allow entry of as many values as [illegible] this way is the part of the program that processes the data after the last variable has been entered.

2. WRITE DOWN THE SOLUTION. Following is a flowchart of a program that will allow entry of any number of variables and then process the accumulations.

This flowchart is a diagram of a simple procedure called a "loop". A loop is used when a part of a program is to be repeated several times before the program goes forward. For example, in a five-step program in which steps 2, 3, and 4 are to be repeated several times, the program should be written so that each time it reaches step 4 it would automatically loop back to step 2 and start again. After some predetermined event, it should jump out of the loop and carry out step 5 (if any). Depending on how the program is written, a loop can be repeated a many time as is necessary.

IN A LOOP, SPECIAL COUNTER AND JUMP INSTRUCTIONS

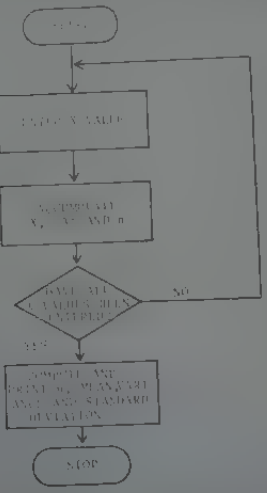


FIGURE 1.

Program Coding Form

CALCULATOR MODEL NO. PROGRAM TITLE

PROGRAMMER

DATE PAGE OF

00	CLEAR ALL	Beginning of data entry loop.	25	AX		50		75	
01	STOP		26			51		76	
02	JRAS		27			52		77	
03	AX		28			53		78	
04			29			54		79	
05			30			55		80	
06			31			56		81	
07	PRINT		32			57		82	
						58		83	
						59		84	
						60		85	
						61		86	
						62		87	
						63		88	
						64		89	
						65		90	
						66		91	
						67		92	
						68		93	
						69		94	
						70		95	
						71		96	
						72		97	
23	PRINT Mean		48			73		98	
24			49			74		99	

SELECT MEMORY 1 2 3 4 5 6 7 8 9

This program will allow us to enter as many X-values as we like. Each time a variable is entered and [RUN] is touched, the program will accumulate ΣX, ΣX² and n, then loop back to step 000. After the last entry is made and [R/S] is touched, the operator merely touches [RUN] without making an entry. This will cause the program to break out of the loop and jump to step 014, where it will process the stored data.

USING THE INDIRECT MEMORY

A common calculating problem is one called "percentage distribution". Several numbers are added together, then each individual number is divided by the sum of all the numbers to determine what percentage each is of the total amount.

Let's see how the problem would be solved without using the indirect memory system. Let's say that we have three numbers, 123, 456 and 789. We would like to write a program that will accumulate a total, print a total and then print the percentage that each number is of the total. We have analyzed the problem in advance and are ready to write down the solution.

Entry	Key Depression	Explanation
	CLEAR REG	To clear all registers
123	+	To add the numbers
	STO IS 01	To store for later use
456	+	To add the numbers
	STO IS 02	To store for later use
789	+	To add the numbers
	STO IS 03	To store for later use

B. Routine to process variables.

```
.xxxHT   01    Recall first entry
         :     Establishes the total as
         :     a constant divisor
         ]
         ;     First percentage
-xxxHT   02    Recall second entry
         ;     Second percentage
-xxxST   03    Recall third entry
         ;     Third percentage
         ;X    Total of percentages
         ;     Number of entries
```

[illegible] approach, so we could not write the [illegible] and [illegible]. However, if we do, [illegible] there are exactly three [illegible]

[illegible], as we did on the previous problem, to handle [illegible], this loop should store the first [illegible] entry in recory 2 and so on. Such a [illegible] for any number of entries, up to the [illegible]. Each entry should be added to the [illegible] stored in a separate memory. After the last [illegible] should automatically leave the loop that [illegible], then enter another loop that recalls the [illegible] to the total to find the percentages. [illegible] the last variable, the program must [illegible] the second loop, go back to step 000 and prepare [illegible] of data.

[illegible] method.

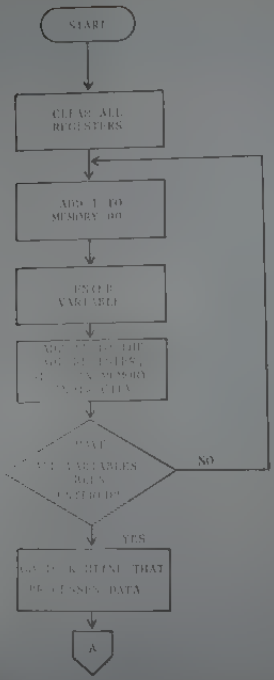

START
CLEAR ALL REGISTERS
ADD 1 TO MEMORY ADD
FETCH VARIABLE
NO
YES
A

FIGURE M

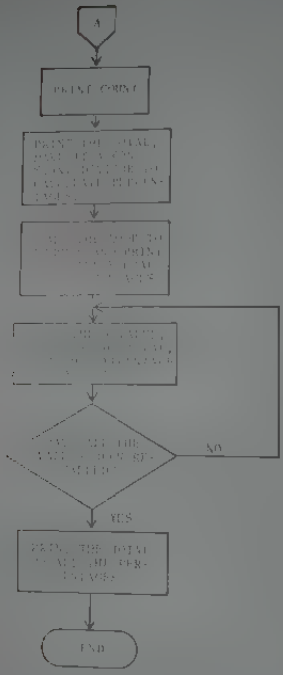


FIGURE N

On two separate occasions, the program must automatically leave a loop. It will be simple to design the program so that it leaves the first loop after the last entry is made. A 'Jump If No Entry' will work perfectly, as it did in the program for standard deviation. We will have a problem with the second loop. The entire loop is automatic and so a 'Jump If No Entry' will not work. To do this we will have to construct an item counter.

The 10B Series has two separate, automatic counters. If you have studied the calculator, you will understand why an item counter (n-count) is used. In the first loop, our program will automatically count the number of entries as we add each entry to the stack machine. All we must do in the second loop is "reverse" the counter, count backwards from the number of entries to zero and then leave the loop when the "reverse count" reaches zero.

The following program uses the indirect register, the reverse counter, and other techniques that have been discussed.

Program Coding Form

CALCULATOR MODEL NO. PROGRAM TITLE

PROGRAMMER

DATE PAGE OF

00	CLEAR	Clear all memories	25		Print number of entries	50	S		75
01	SPACE		26	PRINT		51	JUMP	Jump if more processing required.	76
02	LABEL	Beginning of data entry loop	27	S		52	*		77
03	D		28	PRINT	Print total of entries established	53	LABEL		78
04	Σ+		29	÷	Total is command divisor.	54	D		79
05	STO(+)	Increment M pointer	30	C		55	T		80
06	,		31	-		56	SPACE	Processing completed	81
07	GSUB	Label variable	32	SPACE		57	SPACE		82
08	JUMP		33	N	Initialize counter to skip first loop after last entry.	58	SPACE		83
09	AX		34	+		59	AX		84
10	LABEL	If no entry, jump to process data	35	0	Initialize accumulated time memory	60	Print	Accumulated percentages (proof)	85
11	0		36	STO(+)	Initialize pointer for later use	61	RTN		86
12	1		37	0		62	0		87
13	PRINT	Entry	38	÷	Recall first entry.	63	0		88
14	+	Accumulate variable	39	LABEL		64	0		89
15	RTN		40	0	Data processing	65			90
16	FIXED	Store variables collectively	41	1		66			91
17	JUMP		42	SPACE	Increment pointer	67			92
18	LABEL		43	*		68			93
19	0	Return to beginning of data loop.	44	GSUB(-)	Recall entry	69			94
20	7		45	PRINT	Generate entry's percentage of total	70			95
21	LABEL		46	%		71			96
22	0	Beginning of data processing	47	PRINT		72			97
23	1		48	1		73			98
24	SPACE		49	-		74			99

SELECT MEMORY 1 2 3 4 5 6 7 8 9

Use a paper

This program automatically created a counter by the addition of each variable, then it used the count to form a reverse counter that went back to zero. Also, the program created a constant divisor with the steps [÷] [I] [=]. The answer to this division is of no importance but the total was converted into a constant divisor in only three steps.

As you can see, the indirect memory is a very powerful system. This program took just 65 steps to store and recall as many memories as was necessary. To do this without the use of the "Pointer Register" would have taken considerably more steps.

PRINTING AN IDENTIFIER ON THE TAPE

The Percentage Distribution program will produce a tape that is somewhat difficult to read. The way to solve this problem is to create a counter in the program, and use it to identify the number of each entry and answer. The output may be either of the two following formats.

1.

```
      1.        Identifier
  123.00        (first entry)

      2.        Identifier
  456.00        (second entry)

      3.        Identifier
  789.00        (third entry)

 1368.00        (total of entries)
    3.00        (number of entries)

      1.        Identifier
   8.99         (1st percentage)

      2.        Identifier
  33.33         (2nd percentage)

      3.        Identifier
  57.68         (3rd percentage)

 100.00         (total of percentages)
```

2.

```
      3.        (number of entries)
1368.00         (total of entries)

      1.        Identifier
  123.00        (first entry)
    8.99        (first percentage)

      2.        Identifier
  456.00        (second entry)
  33.33         (second percentage)

      3.        Identifier
  789.00        (third entry)
  57.68         (third percentage)

 100.00         (total of percentages)
```

Either method is acceptable. The first method is simple to program and one that you can write.

The second method is not so obvious. Since each percentage is printed immediately after its corresponding entry, all variables must be entered before the first one can print. The entries will be printed during the second loop, as they are recalled from the memory.

We will have to change the previous program to get identifiers ... tape. A count is already being created in memory ... this count can be recalled and printed during the second ... To make the program print properly, steps 042-047 should ... replaced with steps 042-053 below.

Step No.	Key Depression	Explanation
042	SELECT M	
043	+	Add 1 to memory 00
044	+M/OUT	Recall and print count from memory 00. This will create an identifier on the tape.
045	0	
046	0	
047	SPACE	
048	PRINT	
049	+M/OUT	Recall and print each entry.
050	INDIR	
051	PRINT	
052	%	Compute and print each entry's percentage of the total.
053	PRINT	

... step 043, which is a [PRINT] instruction, should be eliminated ... has been replaced by the [PRINT] instruction at step 051.

Because of the fine editing system available on the 900 Series, you will not have to re-enter this program. If the old program is in the program memory, change it to incorporate the new steps. The old routine was:

Step No.	Key Depression
042	SELECT M
043	*
044	*M/OUT
045	INDIR
046	%
047	PRINT

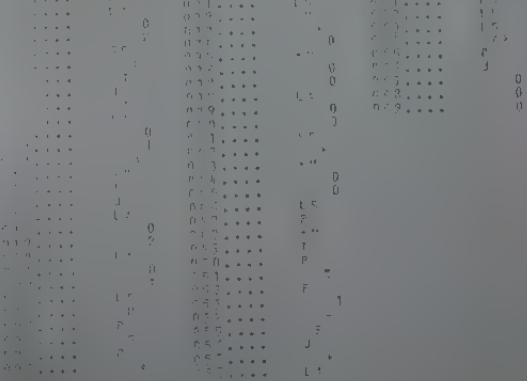
We can leave steps 042, 043 and 044 alone, erase step 045, then insert the new steps from 045 to 051. The program will automatically expand to allow room for the new steps. The [%], [PRINT], instructions that had been steps 046 and 047 will now be steps 052 and 053. Everything else will be adjusted accordingly.

1. Touch [JUMP] 045 [EDIT].
2. Touch [CLEAR] once to erase the [INDIR] instruction.
3. Enter the new steps, [SPACE] [PRINT] [*M/OUT] [INDIR] [PRINT].
4. Touch [MANUAL] [JUMP] 013 [EDIT] [CLEAR] [MANUAL] to eliminate the [PRINT] instruction at step 013.

You might have wanted to correct step 013 before going on to step 014. However, that would complicate matters. When you eliminate step 013, the old step 014 becomes step 013, and so on. Step 045 would become step 044. Therefore it is always best to start with the last change and work backwards.

If you have followed the instructions properly, the tape of the editing instructions should look like this

You may list out the new program by touching [LIST]. The tape should match this one:



To run the program, enter the variables 175, 850 and 750. The printout should look like this:

Headings are extremely important, particularly when writing programs that print a great deal of information.

DEFINITION OF SUBROUTINES

Subroutines are powerful tools that have many uses. If a program has a routine that is to be used in several different places, writing a subroutine can save many program steps. Insert the step [LABEL] [digit] [digit] in front of the subroutine and insert [RETURN] after the last step. (Remember that [digit] [digit] means a two-digit integer such as 01 or 99). Place the subroutine at the end of the main body of the program or in an area that makes sense. The instructions [GO SUB] [digit] [digit] will send the main program to the subroutine, and the [RETURN] instruction at the end will send the program back to the first step after [GO SUB] [digit] [digit].

PROGRAM USING A METRIC CONVERSION SUBROUTINE

The following is an example of a program that uses a subroutine to [illegible] steps. Assume that we must write a program that converts [illegible] to meters and cubic yards to cubic meters. Specifically, [illegible] a number of boxes with measurements given in meters, [illegible] gram should convert each side to its equivalent in yards, [illegible] and the volume of the box in both cubic meters and cubic yards.

[illegible] the volume of a box with length L, width W and height H, the [illegible] is simply; Volume = L x W x H.

[illegible] a number from meters to yards, it should be multiplied [illegible] 1.0936133.

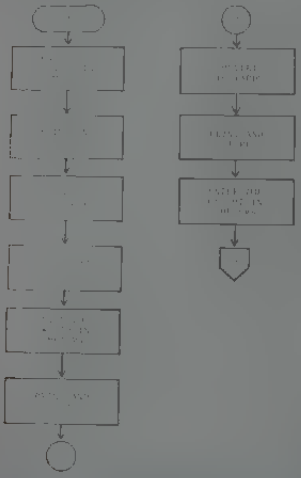
ANALYZE THE PROBLEM

Perhaps the most straightforward method is to enter each variable, store it in meters, convert it to yards, then store the converted number in a separate memory.

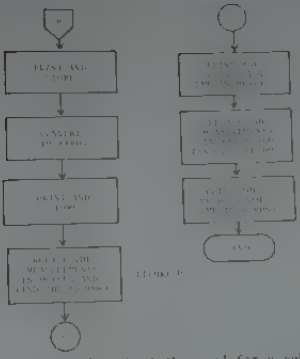
After the third variable is entered, converted and stored, the program can recall the three variables expressed in meters, find their product, print the answer, recall the variables in yards, find the product and print it. Finally, the program should clear all the registers and prepare to accept a new set of data.

2. WRITE DOWN THE SOLUTION.

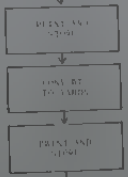
Flowchart the basic logic flow of the program, then incorporate the subroutine technique to shorten the number of steps that the program will use.



FLOWCHART



By studying this flowchart the need for a subroutine becomes obvious. One series of instructions is repeated three times.



The flowchart will be rewritten using the subroutine technique.

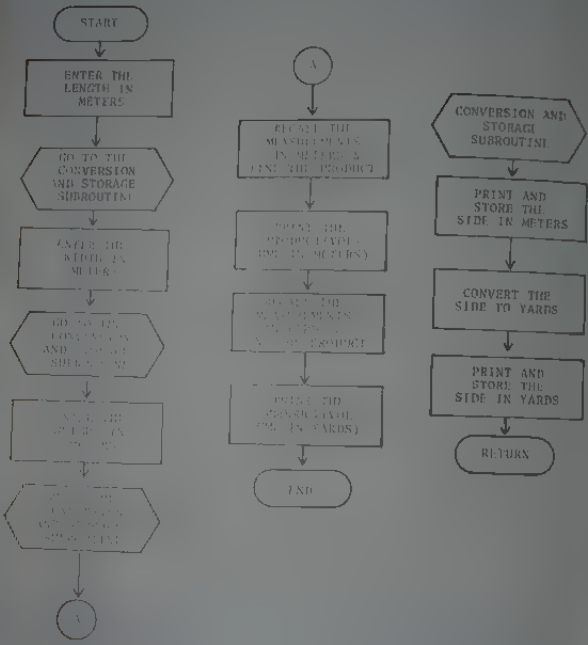

START
ENTER THE LENGTH IN METERS
GO TO THE CONVERSION AND STORAGE SUBROUTINE
ENTER THE WIDTH IN METERS
A
FIND THE MEASUREMENTS IN METERS & FIND THE PRODUCT
PRINT THE PROPERTY LEVEL (IN METERS)
FIND THE MEASUREMENTS IN YARDS & FIND THE PRODUCT
PRINT THE PROPERTY LEVEL (IN YARDS)
END
CONVERSION AND STORAGE SUBROUTINE
PRINT AND STORE THE SIDE IN METERS
CONVERT THE SIDE TO YARDS
PRINT AND STORE THE SIDE IN YARDS
RETURN

FIGURE Q

The flowchart indicates that there are three main routines to be written.

a. Routine to Convert Meters to Yards and Store Both Numbers

Entry	Key Depressions	Explanation
S (meters)	→M/IN 01	(store meters)
	X	
1.093613298577	=	(convert to yards)
	→M/IN 02	

Now write the routine that calculates the volumes. The sub routine is to be used three times.

When a subroutine is used several times to store data that will be used later on in the main program, the Pointer Register could be used to load the memories. (The same logic applies to loops). The subroutine looks like this:

CONVERSION AND STORAGE SUBROUTINE

Key Depression	Explanation
PRINT	Print side in meters
SELECT M	
+	Add 1 to memory 00
→M/IN	
INDIR	Store this side in meters
X	
1	
.	
0	
9	
5	
0	
1	

Key Depression	Explanation
5	
2	
4	
8	
5	
7	
7	
.	Convert this side to yards
SHIFT V	
+	Add 1 to memory 00
STO 1	Store this side in yards
ENTER	

If this routine is used three times, it will load the side measurements into the consecutive registers. If memory 0 is cleared at the beginning of the program, then the first time will go to memory 1, 1 (in yards) will go to memory 2, and the second time will go to memory 3 and 4. Finally, 1 will go to memory 5. Assuming this is the case, write a routine that will find the volume of the box in both cubic meters and cubic yards.

b.

Key Depression	Explanation
•M/OUT	
0	
1	L in meters
X	
•M/OUT	
0	
3	W in meters
X	
•M/OUT	
0	
5	H in meters
•	Volume in meters
•M/OUT	
0	
2	L in yards
X	
•M/OUT	
0	
4	W in yards
X	
•M/OUT	
0	
6	H in yards
•	Volume in yards

Incorporate both routines into one program that allows entry of three varinbles.

INSERT [STOP], [PRINT], [SPACE] AND [JUMP] INSTRUCTIONS.

Program Coding Form

CALCULATOR MODEL NO.

PROGRAM TITLE

PROGRAMMER

DATE

PAGE OF

000	CLEAR ALL	Clear memories	25			50	LBL		
01	STOP	Enter length	26	5		51	0		
02	GTO	SUB	27	=		52	1	Subroutine to convert and store data.	
03	0		28	PRINT	Volume in meters	53	PRINT		
04	1		29	→M/IN	Calculate volume in cards	54	Select M	Increment indirect pointer	
05	STOP	Enter width	30	0		55	.		
06	GTO	SUB	31	2		56	→M/IN	Store varinble	
07	0		32	X		57	INDIR		
08	1		33	RCL		58	X		
09	STOP	Enter Number	34	.		59	1		
10	GTO	Subroutine to convert	35	4		60	,		
11	0		36	5		61	0		
12	1		37	÷		62	0		
13	STO		38	3		63	3		
14	.		39	=		64	6		
15	5		40	3		65	1		
16	2		41	2		66	2		
17	0		42	0		67	9		
18	1		43	3		68	8		
19	RCL		44	6		69	5		
20	5		45	1		70	5		
21	2		46	2		71	=		
22	0		47	9		72	.		
23	1		48	8		73	2		
24	→M/IN		49	5		74	=	Meters converted to yds.	
						75	SELECT M	Increment indirect pointer	
						76	.		
						77	→M/IN		
						78	INDIR	Store varinble	
						79	PRINT		
						80	SPACE		
						81	RETURN	Return to point of departure in main program	
						82			
						83			
						84			
						85			
						86			
						87			
						88			
						89			
						90			
						91			
						92			
						93			
						94			
						95			
						96			
						97			
						98			
						99			

Notice the economy of steps achieved by using a subroutine ([GO SUB] and [RETURN] instructions).

The operating instructions for the program are:

1. Depress [PRGM], [CLEAR], then enter the program.
2. Depress [MANUAL], [RUN].
3. Enter the length in meters and depress [R/S]. The length in yards will print.
4. Enter the width in meters and depress [RUN]. The width in meters and yards will print.
5. Enter the height in meters and depress [RUN]. The height in meters and yards will print.
6. The volume in meters will print, followed by the volume in meters and yards.

   The program is ready to accept a new set of variables.

If you run this program for the variables (5,6,7) and (9.50, 4.41 and 7.71) you will get the following printout with decimal at ll:

5.e — 9.50e
10.38[illegible]

6.e — 4.41e
4.82[illegible]

7.e — 7.71e
5.15091[illegible]

210. — 197.32515
359.0[illegible]

CONDITIONAL SUBROUTINES

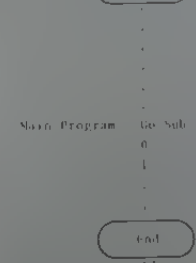
The metric conversion program used an unconditional subroutine. The instructions [GO SUB] 01 sent the program to label 01 under all circumstances. The 900 Series also has the ability to perform conditional subroutines under the same conditions that it will perform conditonal jumps. The calculator tests the number in the keyboard register. It can go to a subroutine under any of the following conditions.

	INSTRUCTION
1. [Is X Positive?]	Branch if Positive [GO SUB] [•]
[Is X Negative?]	Branch if Negative [GO SUB] [-]
[Is X Zero?]	Branch if Zero [GO SUB] [•]
[illegible]	Branch if no Entry [GO SUB] [AX]

[illegible] the program will go to the [illegible] program will pass through [illegible]

NESTED SUBROUTINES

If a program is calculating in a subroutine, it can jump to a second subroutine, go through its steps, return to the point it left off at in the first subroutine, then go back to the point it jumped from in the main program. This process is called "nesting subroutines". The 800 Series can nest subroutines five levels deep, which means it can jump from subroutine A to subroutine B to subroutine C to subroutine D to subroutine E, complete E, go back to D and complete D, then go back to C, B, A and finally return to the main program. The following diagram will explain the concept.



```
                    0
                    2
                    .
                    .
                    .
                    .
                  Return
                    .
                    .
                    .
                    .
                   LA
Subroutine #2      0
                   2
                    .
                    .
                    .
                    .
                    .
                  GO Sub
                    0
                    4
                    .
                    .
                    .
                    .
                  Return
                   LA
Subroutine #3      0
                   3
                    .
                    .
                    .
```

```
                    Go Sub
                    0
                    4
                    .
                    .
                    .
                    Return
                    LA
Subroutine #4       0
                    4
                    .
                    .
                    Go Sub
                    0
                    3
                    .
                    .
                    .
                    Return
                    LA
Subroutine #5       0
                    5
                    .
                    .
                    .
                    Return
```

ERROR CORRECTION SUBROUTINES

The program to compute standard deviation (refer ## loops, page 40) will operate with one exception: what will happen if the operator enters an incorrect value and touches [RUN]? He will have to touch [CLEAR ALL] and start over again. There should be a method of correcting such errors.

This can easily be solved by writing an error-correction routine. Error correction routines are written by following the same three steps that we have been using to write complete programs.

1. WORK OUT THE PROBLEM

   If an incorrect X value was entered and [RUN] was touched, ΣX, ΣX^2 and n would be incorrect. If you had only had a single number, X, then the error correction routine would subtract X from ΣX, X^2 from ΣX^2 and 1 from the n total.

2. WRITE DOWN THE SOLUTION

   The following routine corrects the error and makes it the last one on the tape.

Key Depression	Explanation
	Enter wrong entry X
SPACE	
ENTER	
EXCHANGE	
CHANGE SIGN	
PRINT	Print both X and -X to signify error correction.
CHANGE SIGN	Restore the sign of X
X	Correct error
=	
SPACE	
SPACE	

Because of the spaces and the red printing of x, the tape will clearly show that this is not an entry but an error correction.

The error correction routine can now be incorporated into the main program. To allow the operator to access it manually, label 99 will be inserted at the beginning of the routine. Touching [GO SUB] 99 or [JUMP] [LABEL] 99 [RUN] will take the program to the routine.

The complete program for standard deviation looks like the following one.

```
:  INSERT [STOP], [PRINT], [SPACE] AND [JUMP] INSTRUCTIONS.
```

Program Coding Form

CALCULATOR MODEL NO.

PROGRAMMER

PROGRAM TITLE

DATE

PAGE OF

00	CLEAR ALL	Clear memories	25	AX		50	X
01	STOP		26	=		51	= Nullifies incorrect entry
02	JUMP		27	+		52	SPACE
03	AX		28	=	$\frac{\Sigma X^2 - \frac{(\Sigma X)^2}{n}}{n-1}$	53	SPACE
04	0		29	$\sqrt{}$		54	JUMP
05	1		30	=		55	01 Jump to continue data entries
06	1		31	PRINT Variance		56	01
07	PRINT		32	=		57	1
08	X	Accumulate X, X^2 and n	33	PRINT	Standard deviation	58	
09	+		34	SPACE		59	
10	JUMP		35	SPACE		60	
11	0		36	JUMP		61	
12	0		37	PRINT	Total of data entries	62	
13	1		38	1		63	
14	SUM		39	1		64	
15	X		40	1		65	
16		Error correction routine	41		Error correction routine	66	
17			42			67	
18			43			68	
19			44	SUM		69	
20			45	SUM		70	
21	PRINT	Sum	46	AX	Set all registers	71	
22			47	CMS		72	
23	PRINT	Mean	48	op2=	Multiply identity	73	
24	=		49	STOP	Proper stop	74	

75									
76									
77									
78									
79									
80									
81									
82									
83									
84									
85									
86									
87									
88									
89									
90									
91									
92									
93									
94									
95									
96									
97									
98									
99									

Use additional sheets if necessary

The loading and operating instructions for this program are as follows:

1. Depress [MANUAL] [PROG] [CLEAR]. This brings the step pointer to step 000 and eliminates any old program steps that had been held in the program memory.
2. Enter the program, then touch [MANUAL] [RUN].
3. Enter each X-value.
4. If a mistake is made and [RUN] is touched, do the following:
   a. Touch [GO SUB] 99. The tape will space twice.
   b. The incorrect number will print in red. This will signify that a correction has taken place.
   c. The mistake is now corrected. Continue entering data.
5. After all variables are entered, touch [RUN] to see the following results:
   Count
   Mean
   Variance
   Standard Deviation
6. To find the standard deviation of a new set of data, enter new data.

Rockwell